CS6140: Machine Learning **Fall 2015**

Problem Set 4

Official Due Date: November 24, 2015
Handed Out: November 16, 2015 *Extended Due Date: December 1, 2015*

- Please submit your solutions via your CCIS `github` account.
- Materials associated with this problem set are available at `https://github.ccs.neu.edu/cs6140-02-fall2015/kevinsmall`.
- I encourage you to discuss the homework with other members of the class. The goal of the homework is for you to learn the course material. However, you should write your own solution.
- Please keep your solution brief, clear, and legible. If you are feeling generous, I would *really* appreciate typed solutions (and if you plan on publishing CS/Math/Engineering research, this is actually a good exercise) – see the source materials if you would like to use LaTeX to do this.
- I encourage you to ask questions before class, after class, via email, or the Piazza QA section. However, please do not start e-mailing me questions the night before the homework is due. ☺

Online Learning – 100 points

The purpose of this problem set is to compare the respective performance of a few well-known online learning algorithms by comparing their performance on a synthetic dataset.[1] While the assignments thus far have placed an emphasis on using *real* data, simulations are often better suited for understanding specific properties of learning algorithms.

Specifically, we are going to explore learning a linear function resulting from training on data generated by an l-of-m-of-n function. An l-of-m-of-n is defined over a n-dimensional Boolean vector where there is a defined subset of m attributes such that $f(\mathbf{x}) = 1$ if and only if at least l of these m attributes are active in $\mathbf{x}$. Note that this is a linear function,[2] meaning that Winnow and Perceptron are able to represent the target concept (i.e., the target concept is contained in the hypothesis space).

Algorithms

Specifically, you must implement two online learning algorithms: Perceptron and Winnow (with and without margin in both cases) as defined below for Boolean vectors $\mathbf{x} \in \{0,1\}^n$. Note that these are very closely related algorithms with the most significant different being the update rules.

[1]We will also compare the performance to Naïve Bayes (if you choose for extra credit) since we already have an implementation, which is not an online learning algorithm (although it actually isn't that difficult to code some forms of Naïve Bayes to function in an online fashion).

[2]If you don't see this immediately, spend some time convincing yourself.

Algorithm 1 PERCEPTRON

```
Input: Training data $\mathcal{S}$, Number of rounds $T$, Learning rate $\eta$, Estimated margin $\gamma$

$\mathbf{w} \leftarrow \mathbf{0}$; $b \leftarrow 0$; $t \leftarrow 0$
while $t < T$ do
  for all $(\mathbf{x}, y) \in \mathcal{S}$ do
    if $y(\mathbf{w} \cdot \mathbf{x} + b) \leq \gamma$ then
      $\mathbf{w} \leftarrow \mathbf{w} + \eta y \mathbf{x}$
      $b \leftarrow b + \eta y$
    end if
  end for
  $t \leftarrow t + 1$
end while

Output: Learned hypothesis $\mathbf{w}$
```

Algorithm 2 WINNOW

```
Input: Training data $\mathcal{S}$, Number of rounds $T$, Learning rate $\eta$, Threshold $\theta$, Estimated margin $\gamma$

$\mathbf{w} \leftarrow \mathbf{1}$; $t \leftarrow 0$
while $t < T$ do
  for all $(\mathbf{x}, y) \in \mathcal{S}$ do
    if $y(\mathbf{w} \cdot \mathbf{x} - \theta) \leq \gamma$ then
      $\mathbf{w} \leftarrow \mathbf{w} \circ \eta^{y\mathbf{x}}$
    end if
  end for
  $t \leftarrow t + 1$
end while

Output: Learned hypothesis $\mathbf{w}$
```

Some Notes

While it is difficult for me to preemptively guess which problems you will encounter, here are some notes that I have observed to be confusing for some.

- For PERCEPTRON without margin, if we set $[\mathbf{w}b] = \mathbf{0}$, the learning rate has no effect. Therefore, we will just set $\eta = 1$.
- For PERCEPTRON with margin, the relationship between η and γ are closely related – updates with a large η will more likely make $y(\mathbf{w} \cdot \mathbf{x} + b) > \gamma$. We will set $\gamma = 1$ and experiment with $\eta = \{1.5, 0.25, 0.03, 0.005, 0.001\}$.
- Since l-of-m-of-n functions are monotone, we can use the simplest version of WINNOW as shown, noting that we **do not** update θ. The parameters we will consider includes

$\eta = \{1.1, 1.01, 1.005, 1.0005, 1.0001\}$.

- Winnow with margin is somewhat sensitive. Therefore, while we will experiment with varying $\eta = \{1.1, 1.01, 1.005, 1.0005, 1.0001\}$ and $\gamma = \{2.0, 0.3, 0.04, 0.006, 0.001\}$.

Generating Data

Data is generated as follows. First, the label is generated from a Bernoulli distribution with $\mu = 0.5$.[3]

- For each positive example, select a number of active features from the interval $l \leq a \leq m$. Secondly, select randomly from a uniform distribution a attributes amongst $x_1, \dots, x_m$ and set to 1. Set the other $m - a$ attributes to 0. Set the remainder of the $n - m$ attributes (e.g., $x_{m+1}, \dots, x_n$) to 1 uniformly with a probability of 0.5.
- For each negative example, select a number of active features from the interval $0 \leq a \leq l - 1$. Secondly, select randomly from a uniform distribution a attributes amongst $x_1, \dots, x_m$ and set to 1. Set the other $m - a$ attributes to 0. Set the remainder of the $n - m$ attributes (e.g., $x_{m+1}, \dots, x_n$) to 1 uniformly with a probability of 0.5.

You have been provided with the Java file `GenerateLMN.java` which can be used to generate this data in `LIBSVM` format (sparse representation). Note that we are manipulating $x_1, \dots, x_m$ to represent the relevant portion of the function out of convenience (as there is no loss of generality). However, you should *not* use this information to influence your learning algorithms.

Tuning Parameters

While learning algorithms aren't generally as sensitive as it may seem in this assignment, one purpose of this assignment is to learn how to set hyper-parameters (e.g., η, γ, etc.).[4]

To set the hyper-parameters, we will set aside two distinct subsamples of the training data, $\mathcal{D}_1$ and $\mathcal{D}_2$, each consisting of 10% of the training data. For each set of hyper-parameters settings, training the desired algorithm on $\mathcal{D}_1$ by running the algorithm twenty times over the data. Based on the resulting learned parameters, evaluate this model on $\mathcal{D}_2$ and record the resulting accuracy.

Choosing the set of hyper-parameters that achieve the best performance on $\mathcal{D}_2$, conduct the experiments described in each section. Note that you will be testing the outer product of hyper-parameters settings, meaning that two hyper-parameters with five values each will result in testing 25 sets of hyper-parameters. For your own sanity, I suggest writing code such that this can be automated.

[3]Yes, you too can now make a fair coin flip seem sophisticated.

[4]Note that the process I describe also works for fixed-size data (although there are other reasonable protocols); for this specific problem, we could actually do this differently since we can generate an arbitrary amount of data.

Experiments

1. **[25 points] Counting Mistakes**

   The first set of experiments is to evaluate the specified learning algorithms with two target function configurations: (a) $l = 10, m = 100, n = 500$ and (b) $l = 10, m = 100, n = 1000$.

   In each case, your experiments should consist of the following steps:

   (a) Generate a *clean*[5] 50,000 instance dataset for each specified $\{l, m, n\}$ configuration.

   (b) Fill in the Table 1 with the best performing hyper-parameters:

Algorithm	Hyper-parameters	$n = 500$	$n = 1000$
Perceptron$(\gamma = 0)$			
Perceptron$(\gamma > 0)$			
Winnow$(\gamma = 0)$			
Winnow$(\gamma > 0)$			

Table 1: Tuning Hyper-parameters for Experiment 1

   (c) For each of the four algorithms, run the algorithm over the generated dataset (of 50,00 instances) once and keep track of the number of mistakes, M, that each algorithm makes. Note that a mistake is different than an update (as the margin Perceptron updates for many cases where no mistake is made).

   (d) Plot the cumulative number of mistakes M versus the number of instances $(0 \leq N \leq 50,000)$ observed. For each configuration, plot all four curves on the same graph such that they can be easily compared. In each case, the x-axis should be the number of instances observed, N, and the y-axis should be the number of mistakes made M. You should have two graphs (one for each dataset) with four curves on each. Please clearly label your graphs.[6]

2. **[35 points] Learning Curves**

   The second set of experiments is to construct learning curves. We will begin by setting $l = 10, m = 20$ and vary n such that $40 \leq n \leq 200$ in increments of 40. For each of the 5 different functions, begin by generating 50,000 *clean* instances with the specified $\{l, m, n\}$ configuration.

   As in the previous section, fill in Table 2. To run the experiment,

[5]meaning *not* noisy

[6]If you are having memory issues, you can consider plotting the cumulative error every 10 or 100 instances instead.

Algorithm	Hyper-parameters	$n = 40$	$n = 80$	$n = 120$	$n = 160$	$n = 200$
PERCEPTRON($\gamma = 0$)						
PERCEPTRON($\gamma > 0$)						
WINNOW($\gamma = 0$)						
WINNOW($\gamma > 0$)						

Table 2: Tuning Hyper-parameters for Experiment 2

(a) Present an example to the learning algorithm.

(b) Again, keep track of the number of mistakes, M, the algorithm makes.

The way we will measure convergence is that we will let the algorithm run until S consecutive examples are presented such that no mistakes are made (i.e, the current hypothesis *survives* for 1000 instances). Note that we are able to do this since we know that the algorithms can learn the target function. Once S instances are encountered, record M at this point and halt. For each algorithm, plot a curve of M as a function of n such that the three curves are on the same plot.

Note that you may have to play a bit with the value of S. I would use $S = 1000$ as a good starting point. However, if this is too large such that the algorithm does not halt before $N = 50,000$, you may have to lower this value. However, you must use the same value of S for all experiments for the comparisons to be valid.

3. **[40 points] Batch Performance**

In this case, we will run the online algorithms in batch mode to gain understanding of the relative performance in more common scenarios. These experiments should be conducted as follows:

(a) For a given $\{l, m, n\}$ configuration, generate a *noisy* 50,000 instance training set and *clean* 10,000 instance testing set. We will flip each label with probability 0.05 and each attribute with probability 0.001 using `GenerateLMN.java`.

(b) Optimize the hyper-parameters as in previous sections. Note that technically, since we are running online algorithms in batch mode, T is another hyper-parameter – however, we will just set $T = 20$ as you have probably had enough by now (although if you find something interesting, I would be most interested).

(c) Using these hyper-parameters, train the model with the 50,000 training examples and evaluate your model on the testing data. Report the accuracy of each respective learning algorithm.

Repeat this experiment with the following three $\{l, m, n\}$ configurations.

- $l = 10, m = 100, n = 1000$

- $l = 10, m = 500, n = 1000$
- $l = 10, m = 1000, n = 1000$

For each $\{l, m, n\}$ configuration, the same training and testing data should be used for all appropriate comparisons. This can either be accomplished by using files or setting the random seed appropriately. You should generate results similar to Table 3.

Algorithm	$m = 100$			$m = 500$			$m = 1000$		
	Params	Acc($\mathcal{D}_2$)	Acc(Test)	Params	Acc($\mathcal{D}_2$)	Acc(Test)	Params	Acc($\mathcal{D}_2$)	Acc(Test)
Perc.									
Perc.(γ)									
Winnow									
Winnow(γ)									

Table 3: Tuning Hyper-parameters for Experiment 3

4. **[10 points (Bonus)] Naïve Bayes**
   Repeat the *Batch Performance* experiments with a Naïve Bayes implementation. You may use your own implementation, a publicly available implementation, or the solutions from Problem Set 2 (which will be released before the due date).

What to submit

- A detailed, yet concise report. In addition to the requested information, summarize the findings. Discuss differences in performance of the algorithms and attempt to explain why. Were the results consistent across all experiments? If possible, try to make the report interesting. Note that these experiments were (heavily) influenced by [Kivinen, Warmuth, and Auer; The Perceptron Algorithm versus Winnow: Linear versus Logarithmic Mistake Bounds When Few Input Variables are Relevant, Artificial Intelligence, pg 325-343, 1997] if you would like to do some reading.
- Two plots from the first experiment and one plot from the second experiment. Please clearly label these.
- One table for the third experiment.
- Tables associated with each experiment.
- Source code. Submit all relevant files via `github`. You are free to use the programming language of your choice. However, please include a `README` file that provides instructions on compiling and running your program. Of course, a shell script would be greatly appreciated.